MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE AND NAVIGATION

Approved: David G. Hoag Date: 8 Aug 68
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: D.G. Hoag for Date: 8 Aug 68
R. R. RAGAN, DEPUTY DIRECTOR
INSTRUMENTATION LABORATORY

to be presented at the AIAA Guidance, Control and Flight Dynamics Conference, Pasadena, California, August 12-14, 1968.

E-2307

RECOVERY FROM TRANSIENT FAILURES OF THE APOLLO GUIDANCE COMPUTER

by
Edward M. Copps, Jr.

AUGUST 1968

MIT INSTRUMENTATION LABORATORY

CAMBRIDGE 39, MASSACHUSETTS

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE AND NAVIGATION

Approved: David G. Hoag Date: 8 Aug 68
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: D.G. Hoag for Date: 8 Aug 68
R. R. RAGAN, DEPUTY DIRECTOR
INSTRUMENTATION LABORATORY

to be presented at the AIAA Guidance, Control and Flight Dynamics Conference, Pasadena, California, August 12-14, 1968.

E-2307

RECOVERY FROM TRANSIENT FAILURES OF THE APOLLO GUIDANCE COMPUTER

by
Edward M. Copps, Jr.

AUGUST 1968

MIT INSTRUMENTATION LABORATORY

CAMBRIDGE 39, MASSACHUSETTS

ACKNOWLEDGEMENT

This report was preparedunder DSR Project 55-23870 sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-4065 with the Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, Mass.

RECOVERY FROM TRANSIENT FAILURES OF THE APOLLO GUIDANCE COMPUTER

by

Edward M. Copps, Jr.
Massachusetts Institute of Technology
Cambridge, Massachusetts

ABSTRACT

In the Apollo Guidance Computer, nearly one hundred thousand word transfers occur each second. A random error rate of one in 10^{12} actions would be considered good in today's technology, but at that rate an error might occur within several hundred hours. Added to the random error rate are externally induced errors, including power and signal interface transients, program overloads, and operator errors. The incidence of induced errors has so far been very much greater than random errors (if indeed there have been any at all) in AGC experience.

The AGC is a control computer, and has been designed to detect and to recover from random or induced transient failures. The techniques used are the subject of this paper.

The paper outlines the character of the computer and its system software to a depth sufficient for the discussion. The built-inmalfunction alarm logic is discussed, along with the software-based computer Self Check. These systems, upon detection of a failure, force an involuntary re-ordering of the signal interface, and an involuntary transfer to the restart program logic. From this point, the software, without recourse to suspect information in the central processor, reconstructs the output interface conditions, and reconstitutes the control processes in progress at the time of failure. There are a number of groundrules which limit the recovery capability, based on presumptions about the nature of the failure, These are presented.

The software associated with a restart is described, with a typical program flow derived from an Apollo Mission Program. The amount of memory assigned to restart protection is stated. Several interesting sidelights are briefly discussed such as manual break-in to prevent restart looping, the adoption of the restart technique for scheduling the termination of active programs, and the use of the failure recovery technique to remove a temporary computer overload.

I. Introduction

The Apollo Guidance Computer (AGC) is part of the Guidance, Navigation and Control System of the Apollo Spacecraft. There are two systems, one in the Command Module and one in the Lunar Module. This paper is about the restart feature of the hardware and software. The intent of the paper is to explain what restart protection is, why it is done, what things might have been done instead and implications, large and small, of various design decisions.

The name restart refers to a built-in recovery procedure which responds to hardware detected computer malfunctions. Basically, the hardware transfers control to a specific location in the AGC memory. The software then restarts the programs in progress by reference to stored data which points to the last restart point passed by each program before the malfunction was detected.

II. The Apollo Guidance Computer

The AGC is a control computer. It receives incremental angular and velocity information from an Inertial Measurement Unit, from a telescope, a sextant, and in the case of the Lunar Module it receives range, range rate and angular data from a Rendezvous Radar and a Landing Radar. There is a Display and Keyboard (DSKY) which is illustrated in Figure 1. There is an uptelemetry receiving system, and a downtelemetry system.

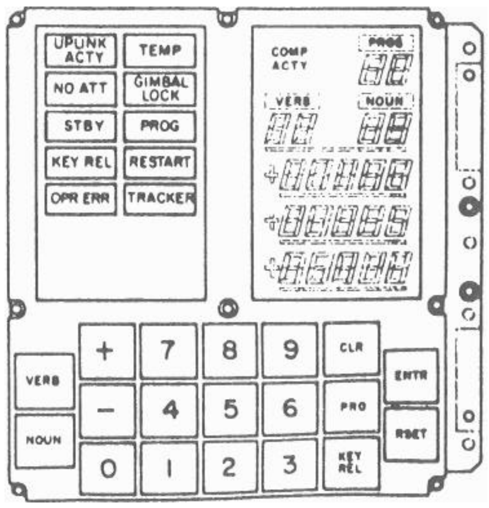


Figure 1 Display and Keyboard Assembly

There are a number of discrete signals sent and received, indicating, for example, such events as liftoff, accelerometer failure, and vehicle stage separation.

The computer directly controls the numerical displays, the rocket engine gimbals, the attitude jet commands, angular commands to the radar, optics, and inertial measurement unit, attitude error displays, engine throttle and engine on/off commands, among other things.

The computer performs many functions, almost all of which are variations on four purposes:

1. to maintain and improve knowledge of position and velocity,
2. to compute changes in positions and velocity by thrust vector control to carry out various mission phases,
3. to control vehicle attitude, and,
4. to compute and display a wide variety of information necessary for astronaut control.

All of these functions are under the control of the astronaut, through the DSKY. There are about 40 separate programs available to him. These programs are function oriented, and are strung together by theastronaut according to his needs by a succession of keyboard actions. In addition to the programs, there are approximately 30 routines which can be run simultaneously with the programs to yield information as desired. In general, only one program can occupy the machine at one time. There is an exception to this which will be discussed in detail later because it is relevant to the restart program.

The progress through a program or a routine is characterized in most cases by a sequence of displays. There are many decision points, at which the relevant data is displayed and a flashing announcement indicates that an astronaut response is required. The usual astronaut response is to Proceed with the computational path and values displayed.

Provision is made to put programs "to sleep" until a desired response or other necessary external event occurs. The display system is a special and intricate part of the AGC software, since display information, possibly requiring an astronaut response, may be "buried" under display information generated by an auxiliary routine, which in turn may have been interrupted by a priority display, or by astronaut keyboard activity. There is a Key Release light which informs the astronaut about buried information.

III. AGC Memory

The AGC memory is composed of three significantly different components. The first consists of 36,864 words of non-alterable program storage, called fixed memory. The contents of the memory locations are fixed at manufacture time. The question of content is one of physical threading or bypassing of wires through a sequence of tape wound magnetic cores. Restart protection presumes that fixed memory is intact.

There are 2040 words of program-alterable memory, called erasable memory. These consist of a conventional destructive read out ferrite core matrix. The question of content is one of the polarity of magnetic flux set in theindividual core. Restart protection presumes that erasable memory is intact, although a small portion is redundantly stored and compared during a restart.

Table 1

OCTAL PSEUDO-ADDRESS	REGISTER NAME	REMARKS	TYPE
00000	A	Accumulator	
00001	L	Lower Accumulator	
00002	Q	Return Address Register	
00003	EB	Erasable Rank Register	Flip-Flop
00004	FB	Fixed Bank Register	Registers
00005	Z	Next Address Register	
00006	BB	Roth Rank Registers	

There are seven Central registers, designed for high speed operation. The names of these registers are in Table 1. These registers are composed of flip flops, formed by interconnecting logic NOR gates, with content dependent on continuity of electrical signal from the time thememoryis set, to the time that it is read. Restart protection presumes that the content of these registers may have been destroyed.

There are sixteen input/output channels, similar to the central registers, employing logic flip flops.

IV. The AGC Software Operating System

There is one central processor. There is an Executive program which schedules Jobs, according to program assigned priority. Each Job is required to periodically interrogate the NEW JOB register which indicates the presence of Jobs of higher priority. If there is a Job of higher priority awaiting execution, the current Job is replaced. At the completion of a Job, Jobs of successively lower priority are executed until finally the Job of lowest priority called DUMMY JOB is continuously executed. A computer Self Test program, designed to test the reading of fixed memory and the reading and writing of erasable memory can be executed during this time. When a Job request is made, a fixed length set of erasable core is assigned to that Job and is kept intact until the Job is terminated. Once a Job is active, it is not possible to recognize it among the other Jobs in various stages of completion in the Executive. This is a difficulty when, as is frequently necessary, a subset of Jobs active in the Executive must be terminated.

V. Interrupt Mode

There are several events, such as a keyboard depression, the overflow of various timing counters, the availability of data at the radar interface, etc., which cause an involuntary transfer of control to specific locations in the AGC. The computer is then operating in interrupt mode. After storing away the information needed to continue the interrupted program, the interrupt program either can complete a short computational task before exiting from the interrupt mode, or it can set up a Job to be done by the Executive after terminating the interrupt.

There are several timing registers which can be set under program control, and which cause interrupts when they come due. These are used to schedule time critical events. An example is TIME 3: when the TIME 3 register overflows, it means that a so-called Task must be executed. This Task is performed under interrupt. The Waitlist program is used to set the TIME 3 counter and to maintain a list of starting addresses for these Tasks. A typical example is the READACCS Task during powered flight. This Task reads the accelerometer velocity increment registers, makes a Job request to the Executive which will complete a rather lengthy series of computations, makes a request to the Waitlist to reschedule itself for 2 seconds from now and then terminates the interrupt mode. When this happens, the Executive resumes where it left off, unless another interrupt has occurred in the meantime. There is an instruction, INHINT, that delays interrupts until execution of another instruction, RELINT.

A restart may be considered to be a special case of interrupt. However this and only this event will interrupt a program already operating in interrupt mode, or a program operating with interrupt inhibited. Also, restart does not return control to the interrupted program.

VI. Need For Restart

In the AGC, nearly one hundred thousand word transfers are performed each second. A random error rate of one in 10^{12} actions would be considered good in today's technology, but at that rate, an error might occur within several hundred hours. Added to the random error rate are externally induced errors, particularly power transients. The possibility of error presents a requirement for a means of recovery, for the AGC itself would still be perfectly useful given that it could be re-initiated into the program. To meet the need for re-initiation, the AGC is equipped with a restart feature comprising alarms to detect malfunction and a standard initiation sequence which leads back into the programs in progress.

VII. Logic Associated With Restart

A restart is caused by a detected computer malfunction. When a restart occurs there is an immediate involuntary transfer of control to location 4000, in fixed memory. Also several signals internal to the AGC are set to insure a known logical state. All the output channels, which include the following functions, are reset (cleared):

a) individual jet commands
b) display relay commands
c) engine off and engine on commands
d) various warning and status lights
e) various moding commands to other spacecraft systems

f) engine gimbal motion commands
g) down telemetry word in process of transmission

Clearing these output channels at least places the interface in a known condition. Otherwise, because of the restart, it is not certain what their status would be. It is left to the software to reconstruct the correct output interface as rapidly as possible.

The coding following the transfer to location 4000 is designed to reinitialize the system programs. Incremental clocks are set to get rapid interrupts to start the various time dependent functions. Jobs in the Executive and Tasks in the Waitlist are cleared out. The display panel is blanked except for the restart light. The various discrete monitors are started. The digital autopilot is initialized. The desired state of the engine is checked and the output bit is properly set.

Following this sequence, the program tests a phase pointer for each of five restart groups. Since there may be several Jobs in progress, running without synchronization, it is necessary to be able to advance restart points in different programs independently. To handle this, there are five independent restart groups, each with its own phase pointer. Jobs and Tasks that may run simultaneously must be in different groups. The pointers are stored redundantly. If each phase pointer passes a redundancy check, the pointers are used to restart the groups.

MAJOR TASKS FOLLOWING A RESTART:

a) Save diagnostic information,
b) Make executive and waitlist dormant,
c) Initialize the system program,
d) Check for lock out,
e) Test erasable,
f) Determine if programs were in process and where to re-enter them.

VIII. Malfunction Detection Devices

There are six malfunction detection devices that cause a restart. They are discussed below.

Parity failure detection is the most powerful. A single odd-parity bit is added to each fixed-memory word at manufacture time, and to each erasable word at write time. Parity is tested at read time. If the test is failed, the restart sequence begins.

The Night Watchman failure detection works as follows: the NEW JOB register, mentioned earlier, must be periodically tested by any program which is following the rules which make the Executive work. This register is “wired” and if it is not tested often enough, the Night Watchman circuit causes a restart.

The TC Trap detects the endless one-instruction loop caused by transferring control to a location L which contains the instruction “transfer control to location L".

The Oscillator Fail is caused by a stopping of the timing oscillator.

The Voltage Fail circuits monitor the 28-, 14-, and 4-volt power levels which drive the computer.

Rupt Lock detects excessive time spent in interrupt mode, or too much time spent between interrupts.

IX. Assignment of Restart Points

At the time of the detected malfunction, the computer may or may not be in interrupt mode. In any case, there was a list of Jobs and a list of Tasks, with associated incremental times, awaiting execution. An uncompleted interrupt mode program is usualiy not restarted, with the exception of Tasks and certain autopilot functions. This means that a downtelemetry word may be garbled, or a key depression may have to be repeated. The restart pointers allow the restart program to pick up proper Job and Task information for a proper restart of the active programs.

Lists of such restart informationare stored in fixed memory.' The lists are called Restart Tables. During the execution of the program, a pointer is advanced to successively point at the proper restart information for that phase of the solution. This is part of the design of the program.

As an alternative to setting a pointer to pick up the information from the proper point in the phase table, other options are available. Often, the instruction that began the last display sequence is an acceptable restart point. This is also a convenient place since the same information relevant to restarting the program must be saved by the Display Program for its own purposes. This type of restart is frequently used and has a special defining format. Another frequently used technique is to define the restart point to be the instruction to be executed next. This does not require use of the Restart Table.

It is occasionally necessary to simultaneously restart combinations of Jobs and Tasks. In other circumstances it is necessary to specify the restart information indirectly, that is, the restart pointer picks up the address where restart information can be found instead of picking up the information directly.

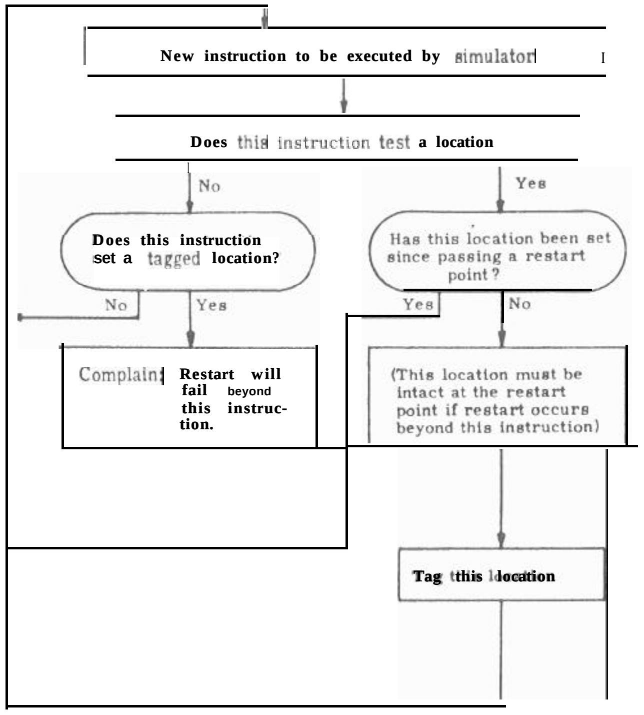
The information for defining a restart point is packed in binary statements which are decoded by a subroutine. The subroutine advances the pointer or performs whatever other operations are needed to make provision for a possible restart. Thus, between invocation of this subroutine, the information about how to restart the software waits, stored in erasable locations, until the mainline programs again use the subroutine to advance the restart information.

X. Testing for Restart Protection

Restart logic is tested during the development of a flight program by causing restarts to happen at specified times during simulations of mission sequences. These simulations are at the instruction level of the AGC and provide detailed checks on program design. Restarts can also be caused during simulation using a real AGC connected to test equipment. These restarts are benign, in the sense that there is no malfunction, and therefore no possibility of destruction of memory between malfunction and detection. There is no ability at present to insert the restart between specified instructions, although this would be a useful test option.

There has been some discussion and preliminary planning of a "Restart Analyzer" which examines the code at simulation time to detect and complain when the execution of the instruction violates restartability rules. There are several problems to implementing such an analyzer; first, the statement of a set of sufficient conditions has not yet been achieved; secondly, the analyzer would greatly increase the run time of a simulation, although it would probably decrease total test time.

Figure 2 is an example of one of the many tests that such an analyzer would have to make.


New instruction to be executed by simulator
Does this instruction test a location
No
Yes
Does this instruction set a tagged location?
Has this location been set since passing a restart point?
No
Yes
Yes
No
Complain: Restart will fail beyond this instruction.
(This location must be intact at the restart point if restart occurs beyond this instruction)
Tag this location

The notation "tested" Includes all operations which use the information In the location. Other logical teats would be required to detect use of central registers without first setting the,,,, setting a restart point when Interrupt is inhibit, and a aet of tests associated with restarting Jabs and Tasks.

Figure 2

XI. Application of Restart Techniques for the Control of Program Sequencing

Rendezvous Navigation uses measured data to update knowledge of relative position and velocity. The information is a combination of angle, range and range rate information from various devices on the spacecraft. This program in the AGC is called P20.

Rendezvous Targeting uses various algorithms to compute velocity changes required to complete the rendezvous. There are several targeting programs, which correspond to different techniques and phases of rendezvous. In all there are six astronaut-callable programs that do Rendezvous Targeting.

During the development of the specifications for these programs, it became apparent that the time spent in the targeting programs should also be used for navigation measurements, Thus a requirement existed for simultaneous running of P20 and one of the various targeting programs, say P34 (Transfer Phase Initiation). It soon became apparent that it was important to be able to start and stop navigation and targeting functions independent of each other.

The mechanization of this simultaneous program capability, including the ability to selectively terminate part of the functions, is similar to the procedures used in recovering from a detected malfunction. When any program is to be terminated, all future Tasks yet to go active are cleared out of the Waitlist. All Jobs whether they are currently active, sleeping, or waiting for their priorityto come to the top, are terminated. Then, the restart group or groups associated with the functions that are to be continued are restarted. That is, they are activated according to the Restart pointer associated with that group.

Another example of the application of this technique occurs during powered flight where, in at least one non-predictable circumstance, all functions, no matter what their stage of completion, have to be terminated, except for the integration of position andvelocity. This is easily implemented by terminating all computations and restarting the restart group associated with integration.

Thus the logic associated with restart protection serves a significant function independent of the question of attempting to recover from detected hardware failures. There are, of course, other design paths that could have been followed to obtain the ability to selectively terminate active programs. It is interesting to note that the Executive system, which does not maintain a record of starting addresses (or other clues as to the origin of the Job) beyond the time when a Job is first started, probably strongly influenced the decision to selectively terminate Jobs by the restart technique.

XII. Software Abort Conditions

Programming dead ends, which are not expected to be encountered in a checked-out program, under proper use, are broken into two categories; the first and largest category involves conditions which probably will recur again and again if the program is restarted. These conditions are handled by abandoning the program in progress and flashing a request to select another program. Also an alarm light is lit and, by keyboard action, the astronaut can find the code number of the alarm that caused the program abort.

There is a small class of problems centering around overloading the Executive or Waitlist, where it is likely that trying the program again via the restart logic will be successful. This is done. A large class of auxiliary programs called Extended Verb Routines can, through astronaut keyboard command, be run simultaneously with a program. If an Executive overload occurs, and if the condition is partially the result of a so-called Extended Verb Routine, restart will solve the problem because, by design, extended verbs are not restartable. However, to differentiate the procedure from hardware malfunctions, the restart warning light is not lit.

XIII. Diagnostic Information

The contents of the program counter and the bank registers, which together point to the instruction that was being executed when a restart occurred, are saved by software means. They may, however, have been destroyed by the malfunction.

It is not possible to differentiate between the various malfunction detectors while the computer is in a flight configuration. There is information brought out through a test connector which permits isolation to a particular failure detection mechanism during ground test.

There is, of course, the possibility that restarts are self-generating. If the same problem still exists on re-executing the code, there is a potential endless loop. In view of this, it would probably be better to store away the diagnostic information only once, on the first pass through the restart logic. There is a register which is incremented on each pass through the restart logic.

XIV. Protection Against Endless Loops

Since the restart logic tries to do a piece of program over again, the probability of endless looping must be dealt with. When this occurs, the loop can be so tight that ordinary means of controlling the computer are not available. For example, the Display Keyboard, or the telemetry input programs, are not serviced. Even turning off the computer does not necessarily break the loop. Therefore, the first thing that is done in the restart logic is to check for a simultaneous depression of two buttons (input keys). If the user depresses these keys, the logic diverts the program to an inert condition, called Fresh Start.

XV. Self Test versus the Restart Philosophy

Self Test tests the ability to read and write into erasable locations. The process consists of storing the contents of an erasable location in another location, writing and reading all zeros, then, writing and reading all ones, then retrieving the original contents and writing back into the register.

Self Test presents an interesting dilemma when juxtaposed against the philosophy of restart protection. There are two reads and two writes (the ones and zeros) that if unsuccessfully completed will be detected. On the other hand, there are two reads and two writes that must be successfully completed at each test in order not to destroy the contents of the register. Self Test cannot itself detect an error in this part of its operation. If a transient failure occurs while Self Check is testing erasable, there is as much chance that the contents of a register will be destroyed as there is that a transient failure will be detected and erasable memory preserved. If there has been a hard failure, self test will detect it, and the contents of the register are not as important as finding the failure. Parity detection enters into this discussion, since at least half, and probably much more than half, of all read and/or **write errors** will cause parity failure, which will itself cause a restart. This does not alter the fundamental point that if transient failures are significant enough to necessitate designing restart logic, then running Self Test has a significant chance of destroying erasable information. Thus searching for hard failures by testing read and write logic may cause undetected destruction of memory due to transient failures. Probably the best thing to do is to do Self Test before beginning significant mission phases, but not otherwise.

XVI. Will Restart Work?

If a computer failure is such that erasable memory information is still intact, and if the failure is indeed transient, then there is great confidence that the restart logic will bring the program through the transient.

Will the failure mechanism be so generous? We do not know. No restart has occurred in flight. Those that have occurred in test have occurred with non-rigorously checked-out test programs and are usually traced to software-invoked TC TRAP failures. This was the method of dealing with program "dead ends" and is no longer used. Little definite information is available.

We can gain some feeling for the problem by considering the mechanism used to detect the malfunction.

If there is a parity failure on reading fixed memory, the chances of recovery are very good. This is because the failure would not destroy the memory, and a subsequent attempt will be successful if the failure mechanism has disappeared. More important, the failure is detected immediately, before other damage is done.

A parity failure on reading an erasable location has far less chance of successful restart because the contents of the erasable is suspect, since the memory is a destructive read system, and the register is refilled with the word that just failed parity with the parity bit corrected.

If the restart is caused by detecting a TC TRAP or a Night Watchman alarm, it is likely that the actual failure has occurred many, many, many instructions ago, and that the program has been running wild for a considerable time. Under these circumstances it is not likely that erasable memory is intact.

If the restart is caused by a detected voltage dropout, there is enough capacitance in the power circuits to keep the central registers intact until the completion of the current memory cycle. Therefore, transfers of information to erasable memory will be completed with good information. Intentional failures of power levels have been tried with successful results, and unplanned voltage dropout is the failure most often experienced in ground checkout. Of course, the oscillator also stops, so that there is an unknown time mismatch between the AGC clocks and real time. There is no way to correct this within the ground rules of restart, but ground tracking can detect and correct this loss of time synchronization.

Thus, the chances of a successful restart vary greatly with the source of the detected failure. There are two failure mechanisms that seem to offer the best chance of successful restart, It is important to note that these two failures could be handled by hardware circuits as an alternative to the approach described in this paper. First, parity failure on reading fixed memory. This could be dealt with by reading the location again, which would be much simpler. Second, voltage transients could be solved by saving flip flop information, then letting the AGC start up where it left off.

XVII. What Should the Astronaut Do if the Restart Light Comes On?

If a restart occurs in flight, the astronaut and the ground will be warned by a restart light on the DSKY. Furthermore, the G&N-Caution Light will come on, and if a sequence of restarts occur in a short interval, the AGC Warning Light will come on, along with a light called the Master Warning Light. The program may or may not give evidence of memory loss. If it does, then erasable memory should be reinitialized with help from the ground. If the program continues to work well, erasable memory should still be initialized at the earliest convenient time. Self Test should be run, and all other means of diagnosis should be called upon. The computer should not be presumed defective although, depending on the flight circumstances, the best strategy might be to go to backup equipment until confidence in the computer can be reestablished.

XVIII. The Cost of Restart Protection

The principle involved in the development of this program is simple. If a detected hardware malfunction occurs, then, since it is highly likely that the failure is transient, it is worthwhile to try to continue the computation. The alternative is to abandon the computer until a new erasable load is transmitted to the ground. The specific techniques developed are complicated and increase the difficulty in delivering checked-out flight software. Some of these effects can be described quantitatively. First, the programs associated with restart occupy 1048 locations of the 36,864 words of fixed memory.

Programs that are restart protected must contain phase update instructions. These occupy 334 locations in fixed memory. Thus about 4% of the memory budget is assigned to restart protection. There is an indirect memory cost associated with designing a program to be restartable.

In terms of testing, some 40% of program testing and 1% of mission phase testing includes restart testing.

There is another cost. Restart protection requires a knowledge of the overall working of the machine, and an intimate knowledge of areas of programthat would otherwise not be of significance in the development of a particular software program region.

XIX. Conclusion

The establishment of restart protection in a computer such as the AGC presents somewhat of an enigma. In a total of over 25 hours of space flight, the computer has yet to have a transient failure from which the restart feature could be called on to demonstrate its worth. This could well be the experience for the whole Apollo program. We have seen that the provision for restart in the computer program complicates the generation and test of program. We have seen that there is a significant class of transient failure events which restart will probably fail to cure. And yet only one successful recovery by restart might save a mission.

REFERENCES

For a general description of the Apollo Guidance Navigation and Control System, see Space Navigation Guidance and Control, Miller, J.E. et al, Technivision Ltd., Maidenhead, England, 1966.

For a detailed description of operational use of the system see Martin and Battin, "Computer Controlled Steering of the Apollo Spacecraft", Journal of Spacecraft and Rockets, vol. 5, no. 4, April 1968.

ACKNOWLEDGEMENT

This report was prepared under DSR Project 55-23870, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-4065 with the Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, Mass.

E-2307

DISTRIBUTION LIST

P. Adler
J. Alekshun
R. Bairnsfather
R. Battin
E. Blanchard
G. Cherry
E. Copps
S. Copps
J. Dahlen
D. DeWolf
G. Edmonds
A. Engel
P. Felleman
J. Fleming
G. Fowks
K. Glick
E. Grace
K. Grenne
M. Hamilton
J. Hand MIT/GAEC
T. Hemker MIT/NAR
J. Henize (5)
D. Hoag
F. Houston
B. Ireland
T. Isaacs
L. B. Johnson
M. Johnston

J. Kernan
K. Kido
*J. Kingston
A. Kosmala
W. Kupfer
A. Laats
L. Larson
R. Larson
T. Lawton MIT/MSC
G. Levine
D. Lickly
F. Little
R. Lones
W. Marscher
F. Martin
R. McKern
H. McOuat
R. Melanson
D. Millard
J. E. Miller
J. S. Miller
J. Morse
J. Nevins
J. O'Connor
J. Parr
P. Philliou
C. Pu
R. Ragan
W. Robertson

P. Rye
J. Saponaro
C. Schulenberg
N. Sears
G. Silver MIT/KSC
B. Sokkappa
W. Stameris
G. Strait MIT/KSC
G. Stubbs
M. Sullivan
J. Suomala
J. Sutherland
W. Tanner
R. Tinkham (3)
M. Trageser
J. Turnbull
J. Vella
K. Vincent
J. Vittek (4)
H. Walsh
R. Weatherbee
R. Werner
R. White
W. Widnall
C. Wieser
M. Womble
R. Woodbury
Apollo Library (2)
MIT/IL Library (6)

*Letter of transmittal only

[illegible]

External:	
NASA/RASPO	(1)
AC Electronics	(3)
Kollsman	(2)
Raytheon	(2)
Capt. M. Jensen (AFSC/MIT)	(1)
MSC:	(21&1R)
National Aeronautics and Space Administration Manned Spacecraft Center Houston, Texas 77058	
ATTN: Apollo Document Distribution Office (PA2)	(18&1R)
C. Frasier	(2)
T. Gibson	(1)
KSC :	(1R)
National Aeronautics and Space Administration J. F. Kennedy Space Center J. F. Kennedy Space Center, Florida 32899 ATTN: Technical Document Control Office	
LRC :	(2)
National Aeronautics and Space Administration Langley Research Center Hampton, Virginia ATTN: Mr. A. T. Mattson	
GAEC:	(3&1R)
Grumman Aircraft Engineering Corporation Data Operations and Services, Plant 25 Bethpage, Long Island, New York ATTN: Mr. E. Stern	
NAR:	(8&1R)
North American Rockwell, Inc. Space and Information Systems Division 12214, Lakewood Boulevard Downey, California ATTN: Apollo Data Requirements Dept. 096-340, Bldg. 3, CA 99	
NAR RASPO:	(1)
NASA Resident Apollo Spacecraft Program Office North American Rockwell, Inc. Space and Information System Division Downey, California 90241	
AC RASPO:	(1)
National Aeronautics and Space Administration Resident Apollo Spacecraft Program Officer Dept. 32-31 AC Electronics Division of General Motors Milwaukee 1, Wisconsin ATTN: Mr. W. Swingle	
GE RASPO:	(1)
NASA Daytona Beach Operations P.O. Box 2500 Daytona Beach Florida 32015 ATTN: Mr. H. Lyman	